FINAL SPRINT

FINISH ALL CODE BY 6/6

TEAM CALENDAR

Final sprint – Game Logic Integration (**Nick**, Darwin, Zhenyu, Katy, Pranay)

- The back of the AI card
- Make sure to pull deck for player from their active deck
- Random pick card for AI
- Fix the animation when same card are put onto the table
- Puppeteer: game loop (win/lose/initial card/draw card)
    - need to mock/fixture randomness
- Jest: game logic - ties, win conditions etc. Code Coverage (~80%)

Final sprint – Game Pages Layout (**Zack**, Jaylynne, Zhenyu, Katy, Haoting (CSS person))

- Home page need to have a background and link to different pages
- Card style on the gamepage
- Exit to site button
- Timer animation
- Winning animation
- Puppeteer: general site navigation (every click = 1+ test)
    - home-page -> game-page -> exit game-page to home-page -> collection-page
- Add color/text/annotation to the card counter
    - Living: "Dining" –blue
    - Dining: "Structure" – Default
    - Structure: "Living"-- Green

Final sprint– Card System & CARD (**ChengCheng**, Ryan, Chung, Destin, Katy)

- CRUD for game collection page for active deck
- CSS for the game collection page
- Fix the styling
- Fix the img ratio
- Replace the black card holder
- Puppeteer: collection page - changing active deck (check CRUD)
- CUSTOM IMG
- Jest: IndexedDB Code Coverage (~80%)

    - UCSDCardsDB init
    - activeDeck - CRUD ops

Guidelines: general rules / reminders for Final Sprint

- General workflow for development:
    - feature branches should be branched off of develop. Make sure to periodically pull from `develop`.
    - Run ESLint and prettier locally:

      ```
      npm run lint
      npm run format
      ```
    - (if applicable) run the unit tests: `npm run test:unit`
    - (if applicable) run the e2e tests: `npm run test:e2e`
    - If you get issues running `npm lint / format, npm ci` should install dependencies
    - When ready to merge to develop, make a PR request. Make sure you pass the CI pipeline. Tag your assigned reviewer at least in your PR.
- Use Github Projects (we will close the previous sprints'). The general project for your team and <FINAL> will be created, but you must create sub-issues and correctly assign yourself to them. Keep your Github Project board updated, and **link PRs by commenting Closes #<issue number>.**
- Write tests in the same branch as the code you are testing.
- Try in to post in #stand-up every other day. If you get off by a day, that's fine. You should have things to put into standup; this is really the only way it will be enforced.
- Use `#breaking-changes` for changes that are made on your end that will affect other teams. Use threads for replies; keep the general channel clear
- Please acknowledge Slack messages, even in `#important-announcements`

Things we need to fix before turn in:

- Naming consistency (repo and variable names within code)
- Clean up branches
- Reorganize repo for clarity and organization
    - Clear entry point of file (README.md), clear starting file name
    - Link to second repo in the main README
- Releases?
- Docs (Readme.md)
- Design related Docs
    - Clean up “Zack’s designs”, wireframes
- UCD related Docs
    - Markdown file of user stories – link in issues
- Architectural related docs
    - Add indexDB init function
    - Flowchart of ci/cd
    - Flowchart of the gameplay
- System related docs
    - CI pipeline diagram ✅
    - How database interacts w/ other components (diagram of how the cards are stored / “where they go”)
- Team & managerial related docs
- Developer Docs
    - JSDoc
    - NEED TO ADD JSDoc COMMENTS FOR FILES OTHER THAN SCRIPT.JS
- Doc location and doc DX
- Sprint planning & estimation
    - We plan to finish our bare-bone at sprint 1, but wasn’t even half way through, etc…
    - We can show that these have improved – sprint planning docs
- Code consistency, commenting, decomposition
    - Linting, commenting: only the script.js has jsdocs (eslint only checks if the comment is valid rather than if it exists)
- Testing
    - Need to up code coverage, pretty well formed pipeline; need to do end user testing / hand testing
- Hand testing (?)

- “Dog Fooding” (?)
    - Yes, we all played it
    - Most of the time, we try it out ourselves before we make a pr
- Stakeholder feedback (?)
    - Weekly TA meetings, especially increased communication towards the end
- Outside User Acceptance Testing (?)
    - Google Form to send out with product and user feedback
- The -ilities (see below)
- Should probably update main with a “working” product → continue
- Branching naming scheme: name-what-you’re-doing
- CI/CD pipeline documentation: cipipline directory has

Ility	**How we implemented**	**Improvements needed**
usability	3 pages, instructions on 2 pages, clear and easy to use nav bar across all pages	How people won in each round (haoting suggested) Improve readability – decrease the card text, perhaps for more information the user can hover over the card to get more information
maintainability	CI pipeline Custom card element Separate navbar code Comments/docs (JSDocs)	Code/file organization More ADRs on CI More JSDoc comments UCD related Docs
scalability	indexDB > localstorage	We might be able to get cloud storage + local storage to increase scalability TrackJS
availability		
reliability	Separate main and develop – always have a working version Reliable retrieval of cards – local read Font fallbacks	Fallback images? Noscript?
portability	Dynamic file paths for localhost vs Github pages rendering	Browser tests?
testability	Unit and e2e testing integrated into the CI	More console logs, errors, warn (Lab 9)

	pipeline	Fix the command for tests in package.json
security	No stored user information	
interoperability	N/A	
performance	Lighthouse tests	Reformat/resizing the image
stability	Many starting points	
accessibility	Lighthouse tests, Error page, meta for SEO	No alt, no enough contrast ratio for background, lists don't only contain , no script
compatibility		Browser tests????

usability - how user-friendly and intuitive the software is

maintainability - how easily the software can be modified to do corrections, improve.

scalability- the ability of the software to handle growing amounts of workload as user increase

availability - the proportion of the time the system is working and functional, measure in percentage

reliability: the ability of the software to perform its required function under stated condition for a amount of time

portability - the ease of the software translate from one environment to another

testability: how effectively the system can be tested for defects

flexibility" how easy the software can be changed in its environment

interoperability: the ability of the software to interact with other system

performance: how the system behaves in terms of RAIL

stability: the software's ability to run without unexpected terminations

: the softwares’ ability to be used by end users with different abilities

Compatibility: the capability of the software to coexist with other system